PA2 Document

Written by Hyeonho Shin, 20203344, Master Course, 010-5136-6003

Summary

My homework is Resnet50-based, which is the well-known Net in image-classification area. But for better operating in our set I used more tricks (Adaptive learning rate, small batch size, modifying ResNet50[1], Kaiming initialization, and Data Augmentation)

The simplest way is to execute **'ForTA.py'**. It loads the weights what I trained using the various technique.

File name	Role	Command
ForTA	Variation of TA code. Scoring program.	Python ForTA.py -i model_file_name.pth
tanukiChar_trainer	Training my Net, and returning model file.	Python tanukiChar_trainer.py
tanukiCharNet	Includes Network structure. I separated it for reducing redundancies.	None. Custom library
tanukiDataAug	Includes custom data augmentation class. In other words, preprocess programs are in it.	None. Custom library

How to use

For scoring

Execute **ForTA.py**. There is no need to use any arguments. It reads **"FtanukiCharNet.pth"** weight file on the same folder.

Where are preprocessing and classifying functions?

All preprocessing steps are included in tanukiDataAug.py. Trainers call them before learning.

It follows Pytorch's transform class. Whenever you read batch data from disk, it preprocesses the images as many as batch size.

In summary, using tanukiChar_trainer.py, what you need to do is…

1. Run tanukiChar_trainer.py
2. Take the model having the best accuracy from step 2.
3. Run ForTA.py for evaluating. If you designate the input file name, type ‘python ForTA.py -i input_file.pth’

Requisites

- Pytorch (1.7.0), (At 1.3.1 version, it will make error because saving method is different.)
- Torchvision
- tanukiCharNet.py, tanukiDataAug.py in same folder.
- Tensorboard (For recording losses and accuracies)
- OpenCV, Pillow(=PIL)

Design

Abstract

For overcoming the small dataset, I adopted 4 techniques.

1. Augmentation.

My codes generate randomly rotated, shifted, resized, and blurred figures without loss. In short, different to the common transform functions, the whole character is not destroyed. It can be achieved by boundary extraction.

My code extracts the ROI (Region of Interest) using bounding box function. To make the function well-operated, the program applies bilateral Filter and adaptive thresholding in prior.

For generating various sample without damage, I resized, shifted, rotated ROI randomly, using LANCZOS for lower resizing noise only in the 350*350 area.

For more variety, I applied random contrast and gaussian blurring also.

Because of the speed problem, I implemented this by two methods. But I attached simpler one, the case of exploiting custom transform function for Pytorch. It is easy to use and more intuitive. (It is implemented in *tanukiChar_trainer.py*)

2. Small batch size

By some papers, they argued that generalization performance increase in more various learning rate under small batch size. It recommends batch size as 8.

3. Adaptive learning rate

At higher learning rate, we can get more faster learning. But there can be divergence or vibration. So, I decrease the learning rate epoch by epoch. I decreased the learning rate as much as 0.99948 times whenever an epoch increases.

4. **Kaiming initialization**

For faster converging, I initialized my net using 'Kaiming initializer'. It has more better speed then one without that because the my net consists of 'ReLU'.

5. Modify ResNet50

Sadly, ResNet50 only supports 224*224 images with 1000 classes. So, I adopted AdaptiveAvgPool2d layer and modified the final FC layer to 5 without downscaling of input images.

Results

Summary

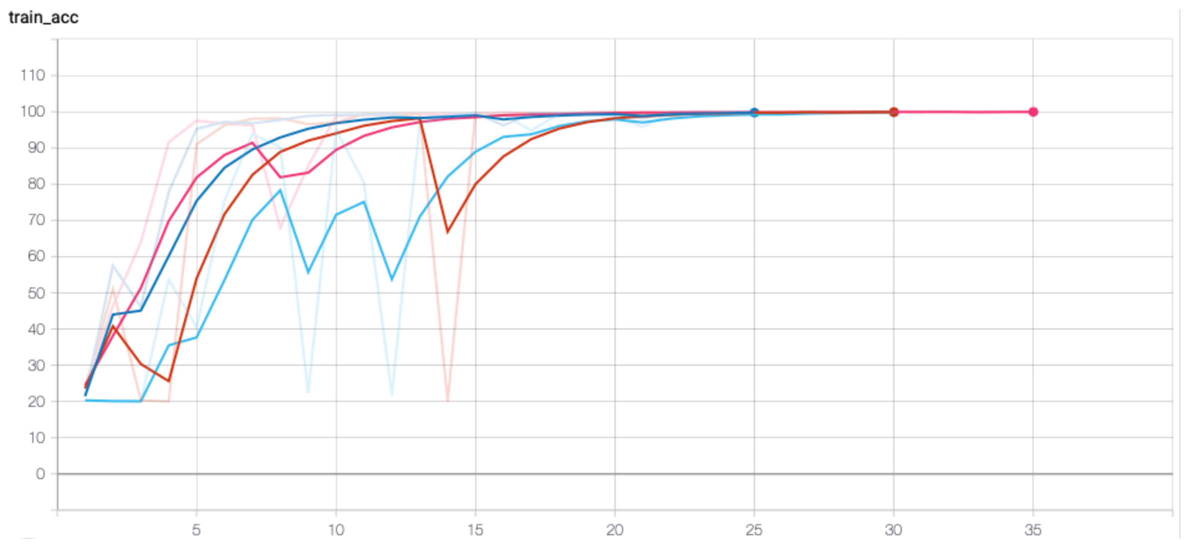
I chose my model with epochs 26, batch size 8, learning rate 0.002. It was the best result in searching space batch size: 8 or 16, learning rate 0.001 or 0.002.

Using my program, I archieved train accuracy 100%, and test accuracy (just I drew) 96%. Of course, the data I drew were never trained.

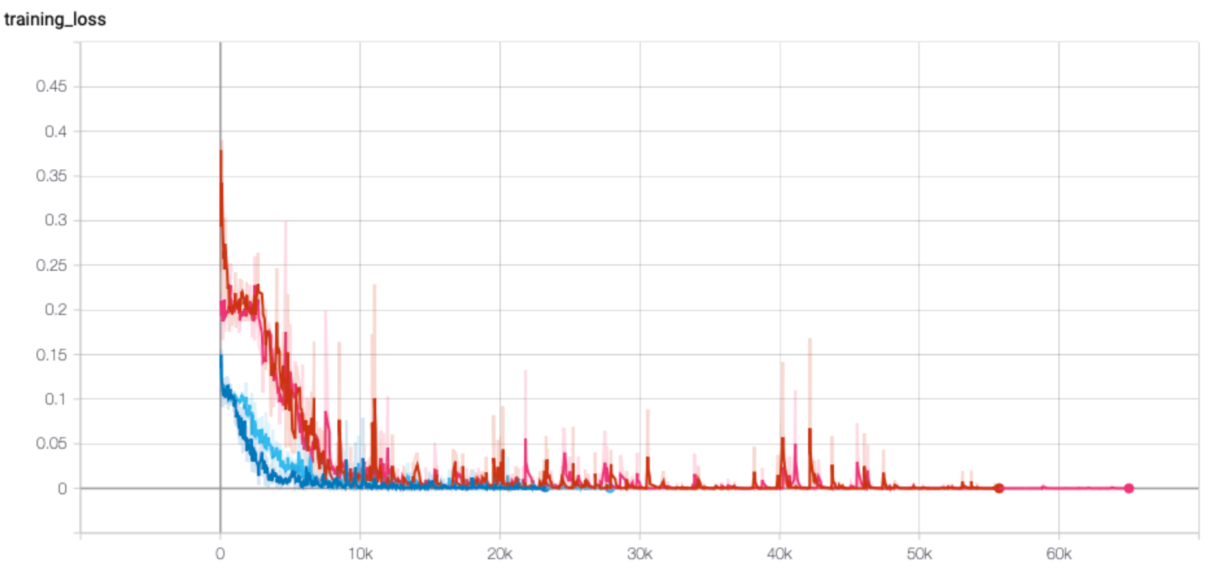
Gray lines are real value, and strong lines are poly-fitted value.

Batch size\Learning rate	0.001	0.002
8	Red	Claret(자홍)
16	Blue	Sky blue

Training Accuracy (by epochs)



Training loss (by iterations)



For faster learning

For faster learning, I separated Preprocessing function as 'gen_aug.py' when training in real. It enhances, take ROI of images and save the results on your disk, './aug_abcde'. So, in this process, all you need is

1. Run gen_aug.py to preprocess.
2. Run tanukiChar_trainer_with_gen_aug.py.
3. Take the model having the best accuracy from step 2.
4. Run ForTA.py for evaluating. If you designate the input file name, type 'python ForTA.py -i input_file.pth'

Because this file has more short epochs, so I modified Learning rate decay speed as 0.95 in this case. The value is detemined by augumenting rate (=100x).

$$0.99948^{\wedge}100 = 0.95 \; \ldots . \; \text{eq.1}$$

I omitted the files because you can be confused of many files. But you can see it at the my git(https://github.com/hyeonhoshin/tanukiCharacters)

REFERENCES

[1] Pytorch's Official ResNet implementation, "ResNet50", https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

[2] Pytorch's Official Tutorial, "모델 저장하기 & 불러오기",

https://tutorials.pytorch.kr/beginner/saving_loading_models.html

[3] Pytorch’s Official Turtorial, “분류기(Classifier) 학습하기”,

https://tutorials.pytorch.kr/beginner/blitz/cifar10_tutorial.html